Sistemi Real Time: formulario

Marco Moschettini

January 15, 2016

1 Problema 1

1.1 Prima di iniziare

- Aggiungere sempre la **colonna** U_i
- Guardare avanti al punto **DMPO** ed aggiungere la **colonna** D_i. (*Se non specificato*, $D_i = T_i$)
- Aggiungere la **colonna** δ_i **con** $\delta_i = \frac{D_i}{T_i}$
- Calcolare U_p come **somma della colonna** U_i

1.2 Clock-driven

- Dimensiona il **ciclo maggiore**: $M = \text{mcm}(T_i)$
- Dimensiona il **ciclo minore**:
  1. $max(C_i) \leq m \leq min(T_i)$
  2. $M \bmod m = 0$
  3. $2m - \text{mcd}(m, T_i) \leq T_i, \forall i$
- Calcolo del **numero dei cicli minori**: $n_{cm} = \frac{M}{m}$
- Calcolo del **numero di job** per processo: $n_{J_i} = \frac{M}{T_i}$
- Identificazione del ciclo minore in cui ciascun job può essere eseguito
  - Tabella composta da $\mathbf{n_{cm}}$ **righe**
  - Tabella composta da $\sum_{\mathbf{i}} \mathbf{n_{J_i}}$ **colonne**
  - Intervalli tra una riga e l'altra pari a m
  - Regole di pianificazione: l'esecuzione del k-esimo job J_{ik} del processo P_i può essere pianificata nel j-esimo ciclo minore c_j se e soltanto se
    1. $(k-1)T_i \leq (j-1)m$
    2. $jm \leq kt_i$
    3. $C_i \leq m - \sum_s C_s,\ \forall s | P_s \in S_j$ con S_j insieme dei processi la cui esecuzione è già stata pianificata in c_j

- Criteri di associazione "job - ciclo minore"
  1. Precedenza ai job di processi con **frequenza di esecuzione più elevata**
  2. Precedenza ai job con **tempo di esecuzione più elevato**
  3. Precedenza ai cicli minori con **minor tempo residuo libero**
- Per inserire una "x" all'interno di una cella della tabella è necessario che l'intervallo di esecuzione del processo (T_i) sia compreso all'interno del C_i. Ad esempio. Ho un processo con $T_i = 15$. Quindi esegue nell'intervallo 0-15, 15-30, 30-45. Se il mio ciclo minore è 10 mi devo chiedere: l'intervallo 0-10 è compreso nell'intervallo 0-15? Sì! quindi inserisco la croce. Il secondo ciclo è 15-30. 10-20 è compreso in 15-30? No, quindi non inserisco la croce. 20-30 invece, è compreso in 15-30? Sì quindi inserisco la croce. E così via...
- Tramite i meccanismi di associazione "job-ciclo minore" del punto precedente compilo la seconda tabella. Se non riesco ad inserire tutti i job allora parte il job slicing
- **Job slicing**:
  * Partiziono il processo con maggior tempo di esecuzione in sottoprocessi
  * Rilasso il vincolo: $P_i(C_i, T_i) \implies P'_j(C'_j, T'_j), P''(C''_j, T''_i)$ con $C'_j + C''_j + \ldots = C_i$ ed i vincoli di precedenza $P'_i < P''_i$

1.3 Priority-driven

- **Test di Han**: $\sum U_i = U_p \leq 1$
- **Test di Kuo-Mok e corollario del teorema di Liu-Layland**:
  - Raggruppo processi con T multipli
  - $U'_j = U_x + U_y + \ldots + U'_z$
  - $T'_j = \min(T_x, T_y, \ldots, T_z)$
  - $C'_j = U'_j T'_j$
  - Liu-Layland afferma che
$$\prod_{j=1}^{N}(1 + U_j) \leq 2$$
- **Test di Burchard**:
  - $X_j = \log_2 T_j - \lfloor \log_2 T_j \rfloor, \ \forall j$
  - $\zeta = \max\limits_{1 \leq j \leq N} X_j - \min\limits_{1 \leq j \leq N} X_j$
  - Condizione di schedulabilità: $U_p \leq U_{RMPO}$
$$U_{RMPO}(N, \zeta) = \begin{cases} (N-1)(2^{\frac{\zeta}{(N-1)}} - 1) + 2^{1-\zeta} - 1 & \zeta < 1 - \frac{1}{N} \\ N(2^{\frac{1}{N}} - 1) & \zeta \geq 1 - \frac{1}{N} \end{cases}$$
- **Algoritmo di Audsley**
  - Calcolo gli R
$$R_i = C_i + I_i(R_i) \leq T_i \quad i = 1, 2, \ldots, N$$

– R_i può essere calcolato nel seguente modo

$$I_i(R_j) = \sum_{j|p_j>p_i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$R_i^0 = C_i$$

$$R_i^n = C_i + I_i(R_i^{n-1}), \quad n = 1, 2, \ldots$$

1.4 DMPO

- **Fattore di utilizzazione efficace del processore**
  - Definisci $p(P_j) = \frac{1}{D_j}$
  - **Ordina la tabella** per $p(P_j)$ crescente
  - H_1: sottoinsieme di processi di priorità $\geq p(P_j)$ con **periodo** $\geq D_j$
  - H_n: sottoinsieme di processi di priorità $\geq p(P_j)$ con **periodo** $\leq D_j$
  - *Suggerimento*: Le prime due celle sono **sempre vuote**.

$$\mathbf{f_j} = \left(\sum_{\mathbf{i} \in \mathbf{H_n}} \frac{\mathbf{C_i}}{\mathbf{T_i}} \right) + \frac{\mathbf{1}}{\mathbf{T_j}} \left(\mathbf{C_j} + \sum_{\mathbf{k} \in \mathbf{H_1}} \mathbf{C_k} \right)$$

  - Verificare se risulta: $\mathbf{f_j} \leq \mathbf{U}(\mathbf{N} = |\mathbf{H_n}| + \mathbf{1}, \delta = \delta_\mathbf{j})\ \forall \mathbf{j}$, (ricorda $|H_n|$ = numero di elementi) con

$$U(N, \delta) = \begin{cases} N((2\delta)^{\frac{1}{N}} - 1) + 1 - \delta, & 0.5 \leq \delta \leq 1 \\ \delta & 0 \leq \delta \leq 0.5 \end{cases}$$

$$\delta = \min_j \{\delta_j\},\ \delta_j = \frac{D_j}{T_j}$$

- *Grafico*
  - Priorità a chi ha D_j **più basso, NON la deadline più vicina!**
  - **Segnare le deadlines**, se si superano allora **fallimento**

1.5 EDF

- **Test basato sulla densitá di utilizzazione del processore**

$$\Delta = \sum_{i=1}^{N} \frac{C_i}{D_j} \leq 1$$

- **Approccio processor demand**
  - Calcolo t^*

$$t^* = \frac{\sum_{i=1}^{N} \left(1 - \frac{D_i}{T_i}\right) C_i}{1 - U}$$

- Calcolo i Busy Intervals BI^n

$$BI^0 = \sum_{i=1}^{N} C_i$$

$$BI^n = \sum_{i=1}^{N} \left\lceil \frac{BI^{n-1}}{T_i} \right\rceil C_i \quad n = 1, 2, \ldots, N$$

  finchè non ne trovo due uguali (vedi Audsley)
- **Scelgo il** $\min(\mathbf{t}^*, \mathbf{BI})$
- Calcolo $D \cap D^*$ come insieme dei D_i minori sia di t^* che di BI
- Calcolo C_p

$$C_p(0,t) = \sum_{i=1}^{N} C_i(0,t) = \sum_{i=1}^{N} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i \leq t$$

2 Problema 2

2.1 Prima di iniziare

- Aggiungere la colonna U_i alla tabella
- Calcolare U_p come somma della colonna U_i

2.2 RMPO

2.2.1 Servizio in background

- Servizio in background posizionato a priorità minore
- Segnare **prima** tutti gli altri processi.
- Quando gli altri processi non eseguono, allora possono essere eseguite le richieste asincrone.

2.2.2 Polling Server

- Regole di riempimento e consumo della capacità di P_s
  1. La capacità è ripristinata al valore C_s all'inizio di **ogni periodo** T_s
  2. La capacità è **progressivamente consumata** durante il servizio di richieste aperiodiche
  3. In assenza di (ulteriori) richieste, la capacità (residua) disponibile è **scartata**
- Consigli
  - Segnare subito l'ingresso delle richieste aperiodiche
  - Come dice la regola 3, se la prima richiesta arriva dopo l'inizio, la capacità è scartata e quindi il server non ripartirà fino al suo T.
  - La capacità si ripristina **sempre** da zero quindi conviene segnarla fin da subito nel grafico della capacità.
  - Il Polling Server non è **mai** ready.

2.2.3 Deferrable Server

- Regole di riempimento e consumo della capacità di P_s
  1. La capacità viene ripristinata al valore C_S all'inizio di **ogni periodo** T_s
  2. La capacità disponibile **è conservata** in assenza di richieste aperiodiche pendenti (il server rimane nello stato *waiting*)
  3. La capacità è **progressivamente consumata** durante il servizio di richieste aperiodiche
- Consigli
  - Solo il server può essere nello stato *waiting*
  - Se è presente capacità, il server si accende non appena arriva una richiesta asincrona (se ha massima priorità ovviamente) e la serve con tutta la capacità necessaria

2.2.4 Priority Exchange Server

- Regole di riempimento e consumo della capacità di P_s
  1. La capacità può essere accumulata, oltre che al livello di priorità $p(P_s)$ proprio di P_s anche al livello di priorità che compete a ciascun processo periodico P_j di priorità $p(P_j) < p(P_s)$
  2. La capacità al livello di priorità $p(P_s)$ è ripristinata al valore C_s all'inizio di **ogni periodo** T_s
  3. La capacità disponibile ad un qualunque livello di priorità **è conservata** durante l'esecuzione di un processo periodico di priorità **non inferiore**
  4. La capacità disponibile al massimo livello di priorità è **progressivamente consumata** sia durante il servizio di **richieste aperiodiche** sia **in assenza di richieste aperiodiche** qualora **non vi siano processi periodici di priorità inferiore pronti per l'esecuzione**
  5. La capacità disponibile al massimo livello di priorità, in assenza di richieste aperiodiche ed in presenza di uno o più processi periodici di **priorità inferiore** pronti per l'esecuzione, è progressivamente trasferita durante l'esecuzione di ciascun processo al corrispondente livello di priorità
  6. In presenza di **richieste aperiodiche**, P_s, se dispone di capacità ad un qualunque livello di priorità, **ha la precedenza** rispetto ad un processo periodico di pari priorità
- Consigli
  - In attesa di richieste aperiodiche, se c'è in giro capacità ad un **qualsiasi livello di priorità**, P_s resta in stato *waiting*
  - Se non è in esecuzione nessun processo, la capacità ad un **qualunque livello viene consumata**
  -

2.2.5 Sporadic Server

- Regole di riempimento e consumo della capacità di P_s

  1. La capacità disponibile è conservata in assenza di richieste aperiodiche pendenti
  2. La capacità è progressivamente consumata **soltanto durante il servizio di richieste aperiodiche**
  3. La capacità viene reintegrata solo dopo essere stata consumata e nella misura in cui è stata effettivamente utilizzata
  4. Indicando con T_A l'istante in cui si verifica la condizione $C_S > 0$ e P_s attivo e con T_D il successivo istante in cui si verifica la condizione $C_s = 0$ e P_s non attivo, l'entità RA (replenishment amount) e l'istante RT (replenishment time) del successivo reintegro sono calcolati come segue

$$RA = \text{capacità consumata in } [T_A, T_D]$$

$$RT = \max(T_A + T_S, T_D)$$

  5. Quando il carico derivante da richieste aperiodiche è tale da **non comportare il totale consumo** della capacità disponibile, le distinte porzioni (chunks) di capacità che ne derivano, devono essere **gestite separatamente**, tenendo traccia, per ciascuna di esse, il corrispondente tempo di reintegro RT. Le regole di reintegro dei singoli chunks sono definite come segue

$$T_E(\text{effective replenishment time}) = \max(RT, t_A)$$

$$RA = \text{capacità consumata in } [t_E, t_D]$$

$$RT = \max(t_E + T_S, t_D)$$

  Due o più chunks sono **consolidati in un unico chunk** quando il loro **tempo efficace di reintegro t_E coincide**.

2.3 EDF

2.3.1 Total Bandwidth Server

- Regole di riempimento e consumo della capacità di P_s

  1. P_s, inizialmente posto nello stato di attesa di richieste aperiodiche, diviene pronto per l'esecuzione non appena si presenta all'istante t_{R_a} una richiesta
  2. Inserita la richiesta in testa alla coda, c_s e d_s vengono calcolati come segue

$$U_s = 1 - U_p$$

$$c_s = C_{R_a}$$

$$d_s = t_{R_a} + \frac{c_s}{U_s}$$

  con c_{R_a} = tempo di servizio della richiesta
  3. P_s viene eseguito in accordo alla priorità dinamica correlata alla deadline d_s, e c_s progressivamente consumata durante l'espletamento del servizio

4. Eventuali richieste che si presentano **prima del completamento del servizio** vengono accodate
5. Terminato il servizio, la richiesta viene rimossa dalla coda
6. **Immediatamente** se una o più richieste sono già pendenti, oppre all'istante di arrivo t_{R_a} di una nuova richiesta, viene identificato il tempo di servizio C_{R_a} della richiesta in testa alla coda e posti

$$c_s = C_{R_a}$$
$$d_s = \max(d_s, t_{R_a}) + \frac{c_s}{U_s}$$

P_s viene dichiarato nuovamente **pronto per l'esecuzione**.

- Consigli
  - Inserire subito la colonna d_{s_i} usando la **regola 2**
  - Il grafico di C_s sarà alto come $\max(C_{Ra_i})$
  - Non segnare i periodi su TBS. Non li avrà
  - Occhio all'ordine! Ordinati per deadline
  - Segnare subito le deadline su TBS
  - Quando è in attesa di richieste aperiodiche il TBS resta in stato di *waiting*
  - Esegue sempre prima il processo con **deadline più vicina**! Nota che nel caso di processi periodici **periodo = deadline**
  - Se esegue un processo con priorità inferiore di TBS mentre questi ha capacità maggiore di zero, TBS resta in stato *ready*
  - Nella tabella sottostante segnare i f_i = (tempi di uscita delle richieste)
- Domanda: **Si verifichi se CUS è in grado di fornire le stesse prestazioni di TBS**
  - Posto f_i = completamento esecuzione
  - Calcolare il tempo di arrivo richiesta a_i e deadline ds_i
  - Se

$$R_{a_i} < ds_i$$

allora, **se TBS prioritario nell'intervallo**

$$f_i(\text{CUS}) = f_i(\text{TBS}) + ds_i - R_{a_i}$$

- Verificare se, indipendentemente dalla effettiva distribuzione temporale, la loro gestione tramite i suddetti server compromette la schedulabilità.
  - Per i processi P_i:
    * disegnare il grafico e verificare (RMPO)
    * Calcolo $C_{S_{\max}}$

$$C_{S_{\max}} = \frac{\text{spazi lasciati liberi dal processo di priorità superiore} - \sum_i C_i * \text{ ripetizioni } P_i}{\text{numero ripetizioni PS}}$$

    * deve valere $C_{s_{\max}} \leq C_s$ (dato nel testo)
  - Per un qualunque altro insieme di processi periodici:

$$U_s = \frac{C_s}{T_s}$$

$$U_s \leq \frac{2}{(1 + \frac{U_p}{N})^N} - 1$$

2.3.2 Constant Utilization Server

- Regole di riempimento e consumo della capacità di P_s

  1. P_s, inizialmente posto nello stato di attesa di richieste aperiodiche, diviene pronto per l'esecuzione non appena si presenta all'istante t_{R_a} una richiesta
  2. Inserita la richiesta in testa alla coda, c_s e d_s vengono calcolati come segue

$$U_s = 1 - U_p$$

$$c_s = C_{R_a}$$

$$d_s = t_{R_a} + \frac{c_s}{U_s}$$

     con c_{R_a} = tempo di servizio della richiesta
  3. P_s viene eseguito in accordo alla priorità dinamica correlata alla deadline d_s, e c_s progressivamente consumata durante l'espletamento del servizio
  4. Eventuali richieste che si presentano prima del completamento del servizio vengono accodate
  5. Eventuali richieste che si presentano **prima di** d_s vengono accodate
  6. Terminato il servizio la richiesta viene rimossa dalla coda
  7. **All'istante** d_s se una o più richieste sono già pendenti, oppure all'istante di arrivo $t_{R_a} > d_s$ di una nuova richiesta, viene identificato il tempo di servizio C_{R_a} della nuova richiesta in testa alla coda e posti

$$c_s = C_{R_a}$$

$$d_s = \max(d_s, t_{R_a}) + \frac{c_s}{U_s}$$

     P_s viene nuovamente dichiarato pronto per l'esecuzione.

3 Problema 3

3.1 Prima di iniziare

- Aggiungere la colonna U_i
- Calcolare U_p come somma di $\sum U_i$
- Aggiungere la colonna ϕ_i degli ingressi dei processi

3.2 DMPO

- **Nota bene**: Tutti gli esercizi devono terminare esattamente nella **stessa unità temporale**! Se ciò non accade significa che è stato commesso un errore

3.2.1 NOP

- Regole

  1. Nessuna regola particolare, segui semplicemente l'attivarsi dei processi.
  2. In caso di risorse annidate, è necessario aspettare che si liberi la risorsa più esterna
  3. Segna un simbolo "**-**" se sei in waiting a causa di un processo di **priorità superiore**
  4. Segna un simbolo "**x**" se sei in waiting a causa di un processo di **priorità inferiore**

3.2.2 PIP

- Regole

  1. La schedulazione dei processi (pronti) è operata in base alle relative priorità correnti. La priorità corrente π_i di un processo $P_j(\forall i)$ coincide con:
     (a) **La corrispondente priorità nominale** p_i nel caso in cui P_i **non detenga alcuna risorsa richiesta da processi di priorità superiore**
     (b) **La più alta fra le priorità correnti dei processi da esso bloccati** in caso contrario
  2. Ad un processo è **negato l'accesso ad una risorsa** solo se essa è già **occupata**. In tal caso la priorità corrente del processo bloccato viene **ereditata** dal processo che detiene la risorsa.
  3. Segna un simbolo "□" quando un processo viene **direttamente bloccato** da un processo di priorità inferiore e cede la sua priorità ad esso.
  4. Segna un simbolo "∘" al di sotto di un simbolo "□" per segnalare che il processo in questione ha subito un **blocco indiretto** dal processo superiore.

3.2.3 PCP

- Regole

  1. La schedulazione di processi (pronti) è operata in base alle relative priorità correnti. La priorità corrente π_i di un processo $P_i(\forall i)$ coincide con:
     (a) La corrispondente priorità nominale p_i nel caso in cui P_i non detenga alcuna risorsa richiesta da processi di priorità superiore
     (b) **La più alta fra le priorità correnti dei processi da esso bloccati** in caso contrario
  2. L'accesso ad una risorsa è **negato** se la priorità corrente del processo che ne fa richiesta **non è maggiore del tetto di priorità del sistema**

$$PC_k(\text{Priority Ceiling}) = \max_k\{p_j | P_j \text{ usa } R_k\} \quad \forall k$$

$$\Pi\Gamma_s = \max_k\{PC_k | R_k \text{ è in uso}\}$$

coincidente con il più elevato tetto di priorità delle risorse al momento allocate, a meno che tale processo detenga già la/le risorsa/risorse il cui tetto di priorità coincide con $\Pi\Gamma_s$.

3. Se ad un processo è negato l'accesso ad una risorsa, la sua priorità corrente viene **ereditata** dal processo che ne ha causato il blocco

- Consigli
  - Segnalare i diversi "Priority Ceiling". Per calcolarli assegnare ad ogni risorsa la priorità massima tra i processi che vi accedono

3.2.4 IPCP

- Regole
  1. La schedulazione di processi (pronti) è operata in base alle relative priorità correnti.
  2. La priorità corrente π_i di un processo $P_i(\forall i)$ coincide con
     (a) La corrispondente priorità nominale p_i nel caso in ci Pi non detenga alcuna risorsa
     (b) **Il massimo tetto di priorità delle risorse in suo possesso** in caso contrario
  3. Processi con la stessa priorità corrente sono schedulati secondo la politica FIFO
- **In pratica**: uguale al PCP ma il processo **non esegue proprio se la sua priorità è sotto al tetto**

3.2.5 Massimo tempo di blocco

- PCP/IPCP
$$B_i = \max\{\text{processi sottostanti}\}$$
- PIP
$$B_i = \min\{\sum \max\{\text{colonna sottostante}\}, \sum \max\{\text{righe sottostanti}\}\}$$

3.2.6 Verificare se i processi sono schedulabili con Audsley

- Seleziona il $\max\{B_i\}$
- Condizione di schedulabilità è
$$R_i = C_i + B_i + I_i(R_i) \leq D_i \quad \forall i$$
$$R_i^0 = C_i + B_i \quad \forall i$$
$$R_i^n = C_i + B_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{n-1}}{T_k} \right\rceil C_k \quad i \neq 1, \quad n = 1, 2, \ldots$$
- Risolvere come un normale Audsley (vedi priority-driven)

3.3 EDF

3.3.1 SRP

- Regole

  1. L'esecuzione di un processo ha luogo solo se:
     (a) **Ha la priorità massima fra i processi pronti e priorità maggiore del processo in esecuzione**
     (b) **Il suo livello di preemption è strettamente maggiore del tetto di preemption del sistema**
  2. Numero colonne = $\max(U_i) + 1$ con U_i unità di ogni risorsa disponibile
  3. Se ho tutte le unità disponibili e tutti i processi possono eseguire allora è ω
  4. Se ho $\nu_k < U_k$ la priorità è la massima tra tutti i processi che **non possono più eseguire avendo quel numero di risorse disponibili** (quelle che richiedono risorse maggiori).
  5. Nel grafico
     - Ogni volta che un processo occupa una risorsa già occupata da un processo precedente controllare il numero di unità disponibile e cambiare il pl (priority level)
     - In caso di risorse innestate considerare sempre il pl della risorsa esterna

- Calcolo del massimo tempo di blocco

  - Si procede come PCP

$$B_i = \max\{\text{processi sottostanti}\}$$

- Analisi di schedulabilità

$$\forall i,\ 1 \leq i \leq N,\ \sum_{k=1}^{i} \frac{C_k}{T_k} + \frac{B_i}{T_i} \leq U_{RMPO}(i) = \begin{cases} 1 & \text{se i periodi sono in relazione armonica} \\ i(2^{\frac{1}{i}} - 1) & \text{in caso contrario} \end{cases}$$